Seminar talk

Java and the Java Memory Model A Unified, Machine-Checked Formalisation

Andreas Lochbihler
Karlsruher Institute für Technologie
ESOP, 2012

Presented by Hartmut Keil

Agenda

- Introduction to the Java Memory Model
- Formalization of Java and its Memory Model
- Proof of the DRF guarantee of the Java Memory Model

What is Memory model?

- A memory model (MM) describes, given a program and an execution trace, whether the execution trace of a program is legal execution of the program.
- Consider the example
  (obviously not correctly synchronized)
- Is the result r1 = 1, r2 = 2 possible?

Initially x := 0 y := 0	
Thread 1	Thread 2
1: r2 = x	3: r1 = y
2: y = 1	4: x = 2

- The result is not possible with *Sequential Consistency*
  (But within the Java Memory Model it is possible)

Sequential Consistency (SC) - definition

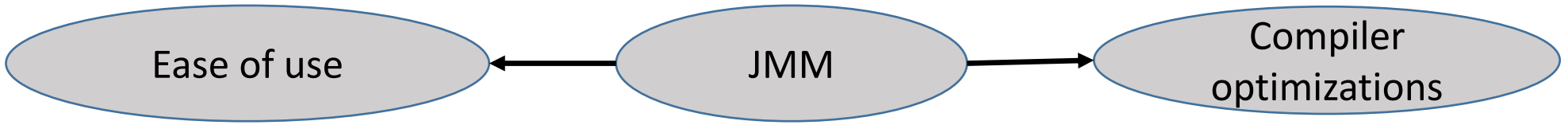
- A execution of a program is sequential consistent, if the result of that execution can be achieved on a uni-processor

$\Rightarrow$ Intuitive interleaving semantic

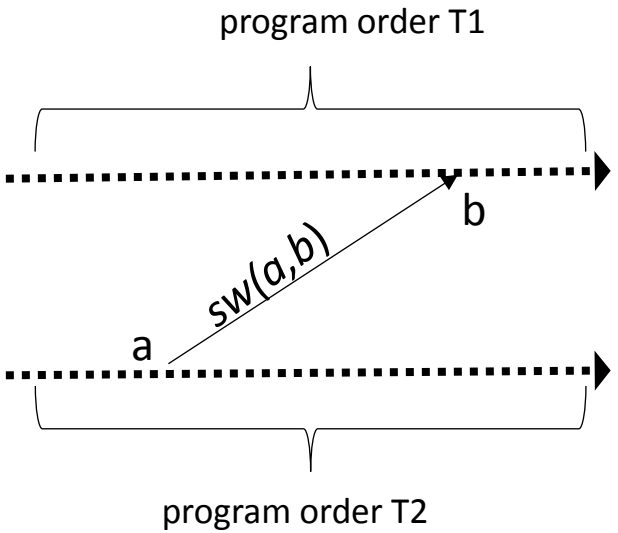
$\Rightarrow$ For a memory model Sequential Consistency disallows most compiler/JVM/hardware optimizations

The Java Memory Model (JMM) - claims

- Sequential Consistency like behaviour for *correctly synchronised* programs (DRF guarantee)
- ‘reasonable’ behaviour for not *correctly synchronised* programs
- Allow as many compiler/JVM optimizations as possible



The Synchronizes-With Order *sw* - definition



Inter-thread actions (not complete)

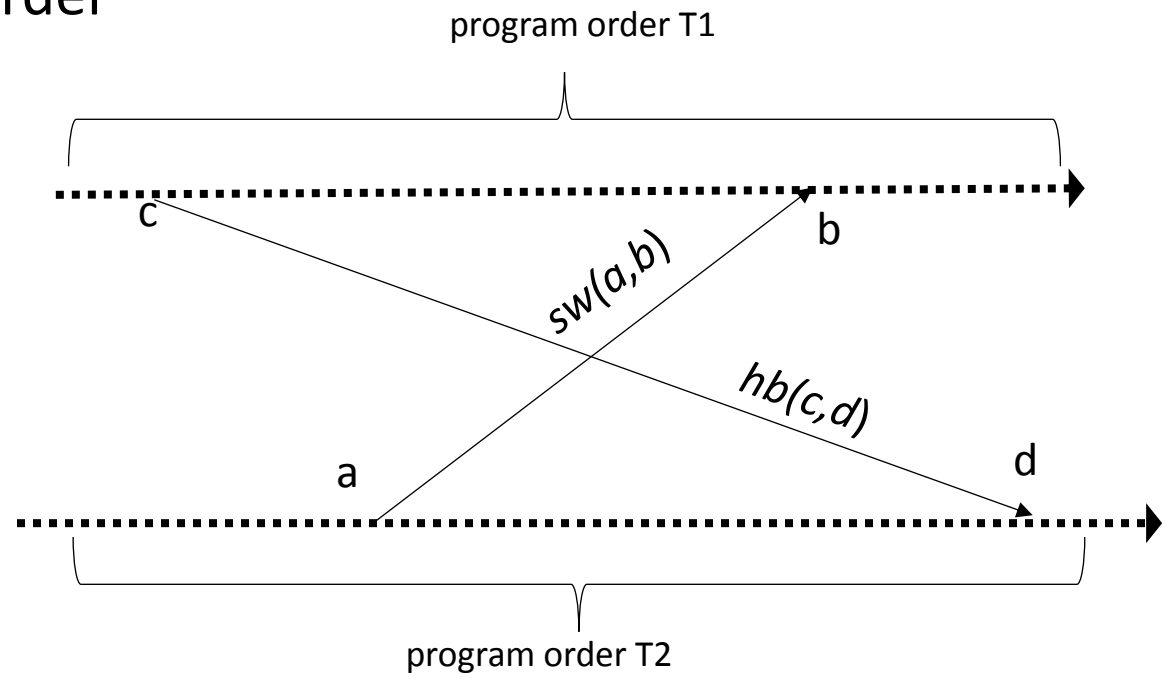
- Read/write of a non-volatile shared variable
- Synchronizations actions:
  - Read/write of a volatile shared variable
  - Lock/unlock of a monitor

Synchronizes-With Order

- An unlock action of a monitor *m* synchronizes-with all subsequent lock actions on the monitor *m*
- A write to a volatile variable *v* synchronizes-with all subsequent reads of *v*
- Notation: *sw(a,b)*

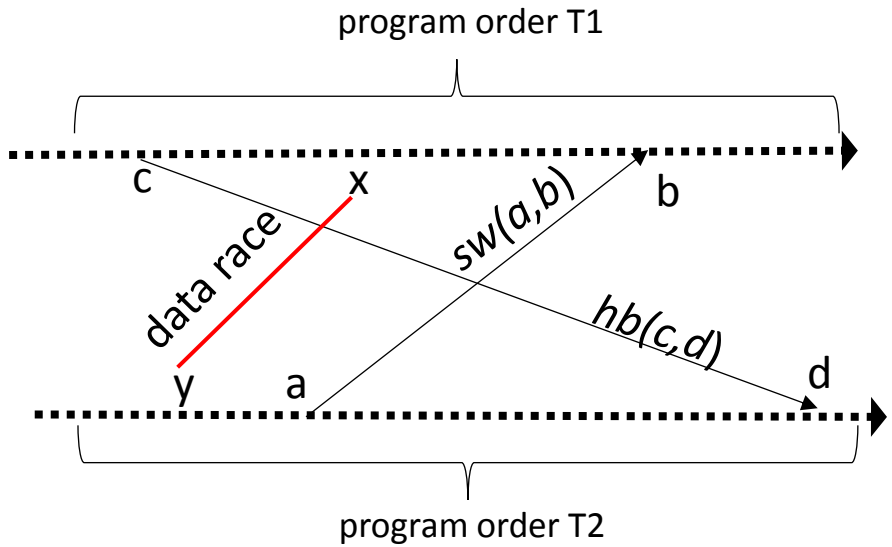
The Happens-Before Order *hb* - definition

- The program order *po* and the Synchronizes-With Order *sw* induces a Happens-Bevor Order
- Happens-Bevor Order is only a partial order
- Notation: *hb(c,d)*



Correctly Synchronized programs - definition

- Data Race:
  - Two accesses *x* and *y* of different threads
  - *x* and *y* are conflicting (at least one is write)
  - No Happens-Before Order for *x* and *y* exists



- Correctly Synchronized program:
  All Sequential Consistent excecutions of the program are free of Data Races

The Java Memory Model – legal executions

- The JMM defines executions of a program *P* as a tuple

$$E = < P, A, po, so, W, V, sw, hb >$$

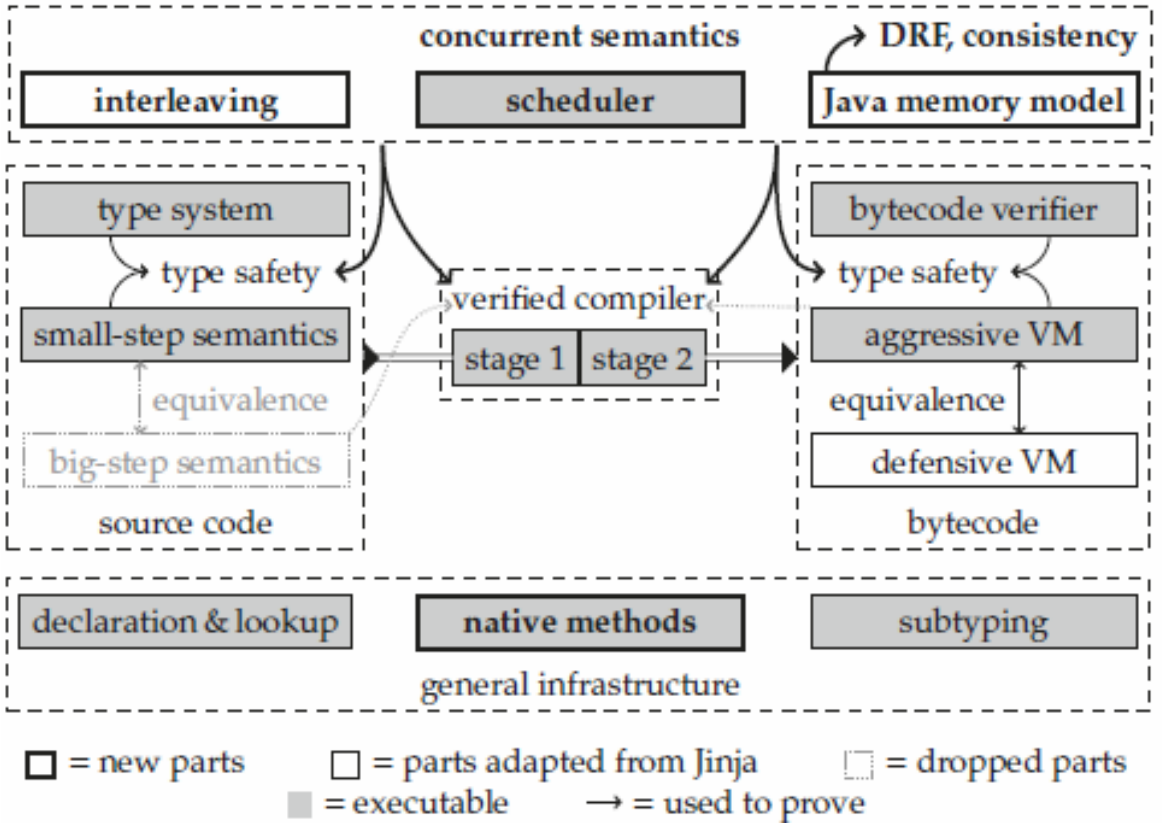
  - set of actions *A* and program order *po*
  - Synchronizes-With Order *sw* and Happens-Before Order *hb*
  - Write-Seen function *W(r)* and Value-Writen function *V(w)*
- A execution is *Well-Formed* if it is *Happens-Before consistent*
  formal: ¬ *hb(r,W(r))*
- Causality requirements for *Well-Formed* executions :
  Every action *a* in a execution must by validated by a well defined *commiting procedure*

The Java Memory Model – formalisation of Java

- The JMM argues in terms of actions and order relationships
- How to connect the JMM to the semantic of Java?
- Idea: formalization of Java

Formalisation of Java - JinjaThreads

- Model of Java-like language
- meta language Isabelle/HOL
- It is a huge model
  - > 20000 lines of Isabelle/HOL text
  - >1000 theorems
- all proves are machine-checked



DRF guarantee – proof

- We have to proof that
  - Given a *correctly synchronised* program
  - All legal executions are *Sequential Consistency*

- Lemma: if every read *r* sees a write *w* and *hb(w,r))* => execution is SC
- Proof
  - assume that a legal execution of a *correctly synchronised* is not SC
  - use the lemma to find an action that can not be commited
  - => contradiction

Future Work

- The formalisation and the DRF proof can be carried over for other languages
- The initialisation specified by the JMM caused complications in several proofs

Questions